First Semester B.E/B.Tech. Degree Examination, Jan./Feb. 2023
Introduction to Python Programming   Subject code: BPLCK105B/BPLCKB105
Scheme and Solutions

Q1 a) What is the need for role of precedence? Illustrate the rules of precedence in Python with example (6 marks)

1a) The concept of precedence is important when it comes to operator evaluation. Precedence determines the order in which operators are evaluated in an expression. It ensures that the expressions are parsed and computed correctly. **(2 Marks)**

The order of operations (also called precedence) of Python math operators is similar to that of mathematics. The ** operator is evaluated first; the *, /, //, and % operators are evaluated next, from left to right; and the+ and - operators are evaluated last (also from left to right). Python will keep evaluating parts of the expression until it becomes a single value. **(2 Marks)**

Example: (2 Marks)

```
(5 - 1) * ((7 + 1) / (3 - 1))
        ↓
        4 * ((7 + 1) / (3 - 1))
        ↓
        4 * (   8   ) / (3 - 1))
        ↓
        4 * (   8   ) / (   2   )
        ↓
        4 * 4.0
        ↓
        16.0
```

1 b) Explain the local and global scope with suitable examples (6 marks)

1b) Parameters and variables that are assigned in a called function are said to exist in that function's local scope. Variables that are assigned outside all functions are said to exist in the global scope. A variable that exists in a local scope is called a local variable, while a variable that exists in the global scopes called a global variable. A variable must be one or the other; it cannot be both local and global. Think of a scope as a container for variables. When a scope is destroyed, all the values stored in the scope's variables are forgotten. There is only one global scope, and it is created when program begins. When your program terminates, the global scope is destroyed, and all its variables are forgotten.

A local scope is created whenever a function is called. Any variables assigned in this function exist within the local scope. When the function returns, the local scope is destroyed, and these variables are forgotten. The next time you call this function, the local variables will not remember the values stored in them from the last time the function was called. **(2 Marks)**

Local Variables Cannot Be Used in the Global Scope

```
def spam():
  eggs = 31337
spam()
print(eggs)
```

The error happens because the eggs variable exists only in the local scope created when spam() is called. Once the program execution returns from spam, that local scope is destroyed, and there is no longer a variable named eggs. So, when program tries to run print (eggs), Python gives an error saying that eggs is not defined. This makes sense if you think about it; when the program execution is in the global scope, no local scopes exist, so there can't be any local variables. This is why only global variables can be used in the global scope.

Local Scopes Cannot Use Variables in Other Local Scopes (2 Marks)

```
  def spam():
❶     eggs = 99
❷     bacon()
❸     print(eggs)

  def bacon():
      ham = 101
❹     eggs = 0

❺ spam()
```

When the program starts, the spam() function is called (5), and a local scope is created. The local variable eggs (1) is set to 99. Then the bacon()function is called (2) , and a second local scope is created. Multiple local scopes can exist at the same time. In this new local scope, the local variable ham is set to 101, and a local variable egg which is different from the one in spam()'s local scope is also created (4)and set to 0.

When bacon() returns, the local scope for that call is destroyed. The program execution continues in the spam() function to print the value of eggs(3) ,and since the local scope for the call to spam() still exists here, the eggs variable is set to 99. This is what the program prints.

Global Variables Can Be Read from a Local Scope (2 Marks)

```
def spam():
  ❶ eggs = 'spam local'
     print(eggs)    # prints 'spam local'

def bacon():
  ❷ eggs = 'bacon local'
     print(eggs)    # prints 'bacon local'
     spam()
     print(eggs)    # prints 'bacon local'

eggs = 'global'
bacon()
print(eggs)         # prints 'global'
```

Output would be:

```
bacon local
spam local
bacon local
global
```

1c) Develop a program to generate Fibonacci sequence of length (N). Read N from the console. (8 marks)

1c) **Code:-** **(8 Marks)**

```
N=int(input("enter the Nth value"))
a=0
b=1
sum=0
i =0
print("Fibonacci series",end=" ")
While(i<n):
        print(sum,end=' ')
       a =b
        b = sum
        sum = a+ b
        i=i+1
```

Output:

Enter the nth value: 5

Fibonacci Series: 0 1 1 2 3

Q2 a) What are functions? Explain Python function with parameters and return statements (7 marks)

2 a) Functions is to group code that gets executed multiple times. Without a function defined, you would have to copy and paste this code each time **(1 Marks)**

- **def Statements with Parameters: (2 Marks)**

When we call the print() or len() function, we pass in values, called arguments in this context, by typing them between the parentheses. We can also define our own functions that accept arguments.

```
def hello(name):
    print('Hello ' + name)

hello('Alice')
hello('Bob')
```

Output:

```
Hello Alice
Hello Bob
```

- **Return Values and return Statements: (2 Marks)**

When creating a function using the def statement, you can specify what the return value should be with a return statement. A return statement consists of the following:

• The return keyword

• The value or expression that the function should return

When an expression is used with a return statement, the return value is what this expression evaluates to. For example, the following program defines a function that returns a different string depending on what number it is passed as an argument.

Example magic8Ball.py: (2 Marks)

```
❶ import random

❷ def getAnswer(answerNumber):
❸     if answerNumber == 1:
           return 'It is certain'
       elif answerNumber == 2:
           return 'It is decidedly so'
       elif answerNumber == 3:
           return 'Yes'
       elif answerNumber == 4:
           return 'Reply hazy try again'
       elif answerNumber == 5:
           return 'Ask again later'
       elif answerNumber == 6:
           return 'Concentrate and ask again'
       elif answerNumber == 7:
           return 'My reply is no'
       elif answerNumber == 8:
           return 'Outlook not so good'
       elif answerNumber == 9:
           return 'Very doubtful'

❹ r = random.randint(1, 9)
❺ fortune = getAnswer(r)
❻ print(fortune)
```

When this program starts, Python first imports the random module (1). Then the getAnswer() function is defined (2). Because the function is being defined (and not called), the execution skips over the code in it. Next, the random.randint() function is called with two arguments, 1 and 9 (4). It evaluates to a random integer between 1 and 9 (including 1 and 9 themselves), and this value is stored in a variable named. The getAnswer() function is called with r as the argument (5). The program execution moves to the top of the getAnswer() function (3), and the value r is stored in a parameter named answerNumber. Then, depending on this value in answerNumber, the function returns one of many possible string values. The program execution returns to the line at the bottom of the program that originally called getAnswer() (5). The returned string is assigned to a variable named fortune, which then gets passed to a print() call 6(and is printed to the screen.

2b) Define exception handling. How exceptions are handled in python? Write a program to solve divide by zero exception (7 marks)

2b) **Exception Handling:**

Right now, getting an error, or exception, in your Python program means the entire program will crash. We don't want this to happen in real-world programs. Instead, we want the program to detect errors, handle them, and then continue to run.

```
def spam(divideBy):
    return 42 / divideBy

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

(2 Marks)

We've defined a function called spam, given it a parameter, and then printed the value of that function with various parameters to see what happens. This is the output we get when run the previous code:

```
21.0
3.5
Traceback (most recent call last):
  File "C:/zeroDivide.py", line 6, in <module>
    print(spam(0))
  File "C:/zeroDivide.py", line 2, in spam
    return 42 / divideBy
ZeroDivisionError: division by zero
```

A ZeroDivisionError happens whenever you try to divide a number by zero. From the line number given in the error message, we know that the return statement in spam () is causing an error.

Errors can be handled with try and except statements. The code that could potentially have an error is put in a try clause. The program execution moves to the start of a following except clause if an error happens. **(2 Marks)**

We can put the previous divide-by-zero code in a try clause and have an except clause contain code to handle what happens when this error occurs**.** **(3 Marks)**

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Output:

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

2 c) Develop a python program to calculate the area of rectangle and triangle print the result. (6 marks)

2c) **Code**

```
def calculate_rectangle_area(length, width):
    return length * width
def calculate_triangle_area(base, height):
    return 0.5 * base * height
# Get input from the user
length = float(input("Enter the length of the rectangle: "))
width = float(input("Enter the width of the rectangle: "))
base = float(input("Enter the base of the triangle: "))
height = float(input("Enter the height of the triangle: "))
# Calculate the area of the rectangle and triangle
rectangle_area = calculate_rectangle_area(length, width)
triangle_area = calculate_triangle_area(base, height)
# Print the results
print("The area of the rectangle is:", rectangle_area)
print("The area of the triangle is:", triangle_area)
```

(6 Marks)

Output:

Enter the length of the rectangle: 2

Enter the width of the rectangle: 3

Enter the base of the triangle: 2

Enter the height of the triangle: 4

The area of the rectangle is: 6.0

The area of the triangle is: 4.0

Q3 a) Explain negative indexing, slicing, index(), append( ), remove( ), pop( ),insert() & sort() with suitable example.(8)

3a) **Negative Indexing:** Negative Indexing is a way of indexing elements in lists and tuples. If a list or tuple has n elements in it then a way of indexing is from 0 to n-1, but in negative indexing, the last element, or the n-1th element is referred as -1, n-2th element as -2 and so on.

E.g., A list containing the first 9 English alphabets in Upper Case

Indexing	0	1	2	3	4	5	6	7	8
Elements	A	B	C	D	E	F	G	H	I
Negative Indexing	-9	-8	-7	-6	-5	-4	-3	-2	-1

(1 Marks)

Slicing: slicing refers to as a selection of a range of elements from a list or tuple. The return type is same as that of the original data container.

Eg : If we have a list a = [1,2,3,4,5,6] and we want to print a slice from 2nd index to 4th (both inclusive) then we do the following

print(a[2,5]) => [3,4,5] **(1 Marks)**

index() : index() functions returns the index of the element, passed in it as a parameter, from a list or tuple. It returns a value error if the element is not present in the container.

Eg : a=[1,2,3,4,5,6]
print(a.index(3)) => 2 **(1 Marks)**

append() : is used to add or append a value in a list or tuple, not altering any value present within the containers.

Eg : a = [1,2,3,4,5,6]

a.append(7)

print(a) => [1,2,3,4,5,6,7] **(1 Marks)**

remove() : is a function which is used to remove a value(passed as parameter) from a list. If the value is not present, a value error is returned.

Eg : a = [1,2,3,4,5,6]
a.remove(5)
print(a) => [1,2,3,4,6] **(1 Marks)**

pop() : pop() function is used to remove an element from a list whose index is passed as a parameter. If nothing is passed, then the last element in the list is removed.

Eg : a = [1,2,3,4,5,6]
a.pop(5)
print(a) => [1,2,3,4,5] **(1 Marks)**

insert() : insert() function is used to insert an element in a list or tuple. It takes 2 parameters which are the index and item.

Eg : a = [1,2,3,4,5,6]
a.insert(2,20)
print(a) => [1,2,20,3,4,5,6] **(1 Marks)**

sort() : it is a function that sorts a list in ascending or descending order. It generally sorts in ascending order and to do it in descending, we need to set the *reverse* parameter in the function to True, which by default is False.

Eg : a = [7,8,9,1,2,3,4,5,6]
a.sort()
print(a) => [1,2,3,4,5,6,7,8,9]
a.sort(reverse=True)
print(a)=>[9,8,7,6,5,4,3,2,1] **(1 Marks)**

Q3b) Explain the use of in and not in operators. in list with suitable examples.(6)

3b) Python "in" operator is used to check whether a specified value is a constituent element of a sequence like string, array, list, tuple, etc. When used in a condition, the statement returns a Boolean result evaluating either True or False. When the specified value is **found** inside the sequence, the statement returns True. Whereas when it is **not found**, we get a False**. (1 Marks)**

Eg:

list1= [1,2,3,4,5]
string1= "My name is AskPython"
tuple1=(11,22,33,44)
print(5 in list1) => True
print("is" in string1) => True
print(88 in tuple1) => False **(2 Marks)**

The “not in” operator in Python works exactly the opposite way as the “in” operator. It also checks the presence of a specified value inside a given sequence but its return values are totally opposite to that of the “in” operator.

When used in a condition with the specified value present inside the sequence, the statement returns False. Whereas when it is not, we get a True. **(1 Marks)**

Eg:

```
list1= [1,2,3,4,5]
string1= "My name is AskPython"
tuple1=(11,22,33,44)

print(5 not in list1) => False
print("is" not in string1) => False
print(88 not in tuple1) => True
```
(2 Marks)

Q3 c) Develop a program to find mean, variance and standard deviation. (6)

3c)

```
a=[1,2,3,4,5,6,7,8,9]
sd=[]
mean = sum(a)/len(a)
print("Mean : ", mean)  => Mean : 5.0
for x in a:
    sd.append((x-mean)**2)

V = sum(sd)/len(sd)
print("Variance : ", V) => Variance : 6.6666666666667
print("Standard Deviation : ", V**0.5) => Standard Deviation : 2.58198889747
```
(6 Marks)

Q4 a) Explain the following methods in lists with an example:

i) len() ii) sum( ) iii) max(0 iv) min( ) (8)

4a) **len() :** it returns the length of the passed string or data container. Its return type is that of an integer.

Eg: a=[1,2,3,4,5,6,7,8,9]
print(len(a)) => 9

sum(): it returns the sum of all the elements(numbers) of a list or tuple, passed as a parameter.

Eg: a=[1,2,3,4,5,6,7,8,9]
print(sum(a)) =>45

max() : The max() function returns the item with the highest value, or the item with the highest value in an iterable. If the elements are string then it will return the string with the maximum lexicographic value

Eg:

var1 = 1

var2 = 8

var3 = 2

max_val = max(var1, var2, var3)
print(max_val) => 8

var1 = "geeks"
var2 = "for"
var3 = "geek"

max_val = max(var1, var2, var3)
print(max_val) => "geeks"

min(): min() function returns the smallest of the values or the smallest item in an iterable passed as its parameter.

Eg :
print(min([1, 2, 3])) => 1
print(min({'a': 1, 'b': 2, 'c': 3})) => 'a'
print(min((7, 9, 11))) => 7

4b) Explain set() and setdefault( ) method in a dictionary.

4b) set() method is used to convert any of the iterable to sequence of iterable elements with distinct elements, commonly called Set. **[3 Marks]**

```
lis1 = [ 3, 4, 1, 4, 5 ]
tup1 = (3, 4, 1, 4, 5)
print("The list after conversion is : " + str(set(lis1)))
print("The tuple after conversion is : " + str(set(tup1)))
```

Output:

```
The list after conversion is : {1, 3, 4, 5}

The tuple after conversion is : {1, 3, 4, 5}
```

The **setdefault()** method returns the value of the item with the specified key. If the key does not exist, insert the key, with the specified value. **. [3 Marks]**

Eg:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.setdefault("color", "white")
print(x) => White

d = {'a': 97, 'b': 98, 'c': 99, 'd': 100}
d.setdefault('g', 32)
d.setdefault('a', 50)
print(d) => {'a': 50, 'b': 98, 'c': 99, 'd': 100, 'g': 32}
```

4c) Develop a Python program to swap cases of a given string . [6 Marks]

Input: Java

output: jAVAa="Java"

4c)

```
for x in a:
    if x.isupper():
        print(x.lower(), end="")
    elif x.islower():
        print(x.upper(), end="")
```

Output => jAVA

5a) Explain join( ) and split ( ) method with examples.(8)

5a) The join() method is useful when you have a list of strings that need to be joined together into
a single string value. The join() method is called on a string, gets passed a list of strings, and returns a string. The returned string is the concatenation of each string in the passed-in list. **(2 Marks)**

Example: **(2 Marks)**

>>> ‘, ‘.join([‘cats’, ‘rats’, ‘bats’])
‘cats, rats, bats’

The split() method called on a string value and returns a list of strings. You can pass a delimiter string to the split () method to specify a different string to split upon. **(2 Marks)**

Example: **(2 Marks)**

>>> ‘MyABCnameABCisABCSimon’.split(‘ABC’)
[‘My’, ‘name’, ‘is’, ‘Simon’]

5 b) Explain with examples: i) isalpha( ) ii) isalnum( ) iii) isspace(). (6)

5b) isalpha() returns True if the string consists only of letters and is not blank**.(1 Mark)**
isalnum() returns True if the string consists only of letters and numbers and is not blank**.(1 Mark)**
isspace() returns True if the string consists only of spaces, tabs, and new- lines and is not blank**.(1 Mark)**

Example : **(3 Marks)**

```
while True:
  print('Enter your name:')
  age = input()
  if age.isalpha():
    break
  print('Please enter alphabets for name')
while True:
  print('Select a new password (letters and numbers only):')
  password = input()
  if password.isalnum():
    break
  print('Passwords can only have letters and numbers.')
```

OUTPUT:
Enter your name:
123
Please enter alphabets for name
Enter your name:
vijay
Select a new password (letters and numbers only):
vijay_123
Passwords can only have letters and numbers.

Select a new password (letters and numbers only):
vijay123

5 c) **Develop a python code to determine whether the given string is a palindrome or not a palindrome**. (6)

```
str_1 = input ("Enter the string to check if it is a palindrome: ")
str_1 = str_1.casefold ()
rev_str = reversed (str_1)
if list (str_1) == list (rev_str):
        print ("The string is a palindrome.")
else:
        print ("The string is not a palindrome.")
```

OUTPUT:
Enter the string to check if it is a palindrome: ABa
The string is a palindrome.

6. a) Explain the concept of file handling. Also explain reading and writing process with suitable example. (8)

6a) Before performing any operation on the file like reading or writing, first, we have to open that file. For this, we should use Python's inbuilt function open () but at the time of opening, we have to specify the mode. The following mode is supported:

r: open an existing file for a read operation.
w: open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.
a: open an existing file for append operation. It won't override existing data.
r+: To read and write data into the file. The previous data in the file will be overridden.
w+: To write and read data. It will override existing data.
a+: To append and read data from the file. It won't override existing data. **(2 Marks)**

<file variable> = open(*<file name>,* "r")

Reading: If you want to read the entire contents of a file as a string value, use the File object's read() method. Alternatively, you can use the readlines() method to get a *list* of string values from the file, one string for each line of text.**(2 Marks)**

Writing: Write mode will overwrite the existing file and start from scratch, just like when you overwrite a variable's value with a new value. Pass 'w' as the second argument to open() to open the file in write mode. Append mode, on the other hand, will append text to the end of the existing file. **(2 Marks)**

Example: (2 Marks)

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello, world!\n')
13
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
Hello, world!
Bacon is not a vegetable.
```

6 b) Explain the concept of file path. Also discuss absolute and relative file path. (6)

6b) A file has two key properties: a filename (usually written as one word) and a path. The path specifies the location of a file on the computer.

Path() will return a string with a file path using the correct path separators. **(2 Marks)**

```
>>> from pathlib import Path
>>> Path('spam', 'bacon', 'eggs')
WindowsPath('spam/bacon/eggs')
```

An absolute path, which always begins with the root folder. os.path.abspath(path) will return a string of the absolute path of the argument. os.path.isabs(path) will return True if the argument is an absolute path and False if it is a relative path.

```
>>> os.path.abspath('.')
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37'
>>> os.path.abspath('.\\Scripts')
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37\\Scripts'
```

(2 Marks)

A relative path, which is relative to the program's current working directory. os.path.relpath(path, start) will return a string of a relative path from the start path to path. If start is not provided, the current working directory is used as the start path.

```
>>> os.path.relpath('C:\\Windows', 'C:\\')
'Windows'
```

```
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
'..\\..\\Windows'
```

When the relative path is within the same parent folder as the path, but is within subfolders of a different path, such as 'C:\\Windows' and 'C:\\spam\\eggs', you can use the “dot-dot” notation to return to the parent folder**. (2 Marks)**

6 c) Briefly explain saving variables with shelve module. (6)

6c) We can save variables in your Python programs to binary shelf files using the shelve module. For example, if you ran a program and entered some configuration settings, you could save those settings to a shelf file and then have the program load them the next time it is run.

To read and write data using the shelve module, first import shelve. Call shelve.open() and pass it a filename, and then store the returned shelf value in a variable. You can make changes to the shelf value as if it were a dictionary. When all are done, call close() on the shelf value. **(2 Marks)**

Example : (4 Marks)

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

We open the shelf files to check that our data was stored correctly.
Entering shelfFile['cats'] returns the same list that we stored earlier, so we know that the list is correctly stored, and we call close().

```
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['cats']
['Zophie', 'Pooka', 'Simon']
>>> shelfFile.close()
```

Just like dictionaries, shelf values have keys() and values() methods that will return list-like values of the keys and values in the shelf

```
>>> shelfFile = shelve.open('mydata')
>>> list(shelfFile.keys())
```

['cats']
>>> list(shelfFile.values())
[['Zophie', 'Pooka', 'Simon']]
>>> shelfFile.close()

7.Explain the following file operations in Python with suitable example: (6 Marks)

i)Copying files and folders

i) The shutil (or shell utilities) module has functions to let you copy, move, rename, and delete files in your Python programs. To use the shutil functions, you will first need to use import shutil.

The shutil module provides functions for copying files, as well as entire folders. Calling shutil.copy(source, destination) will copy the file at the path source to the folder at the path destination. (Both source and destination are strings.) If destination is a filename, it will be used as the new name of the copied file. This function returns a string of the path of the copied file.

Code:-

import shutil, os

os.chdir('C:\\')

shutil.copy('C:\\spam.txt', 'C:\\delicious')

'C:\\delicious\\spam.txt'

shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt')

'C:\\delicious\\eggs2.txt' **(2 Marks)**

ii)Moving file & Folders

ii) Calling shutil.move(source, destination) will move the file or folder at the path source to the path destination and will return a string of the absolute path of the new location. If destination points to a folder, the source file gets moved into destination and keeps its current filename.

Code:-

import shutil

shutil.move('C:\\bacon.txt', 'C:\\eggs')

'C:\\eggs\\bacon.txt' **(2 Marks)**

iii)Permanently deleting file & Folders

iii) You can delete a single file or a single empty folder with functions in the os module, whereas to delete a folder and all of its contents, you use the shutil module.

• Calling os.unlink(path) will delete the file at path.

• Calling os.rmdir(path) will delete the folder at path. This folder must be

empty of any files or folders.

• Calling shutil.rmtree(path) will remove the folder at path, and all files

and folders it contains will also be deleted.

Code:-

```
import os

for filename in os.listdir():

if filename.endswith('.rxt'):

os.unlink(filename)
```

(2 Marks)

7.b) List out the benefits of compressing file? Also explain reading of a zip file (8Marks)

7.b) In Python Zipfile is an archive file format and a compression standard; it is a single file that holds compressed files. Compressing a file reduces its size, which is useful when transferring it over the Internet. And since a ZIP file can also contain multiple files and subfolders, it's a handy way to package several files into one. This single file, called an archive file, can then be, say, attached to an email.

Python programs can both create and open (or extract) ZIP files using functions in the zipfile module.

Python Zipfile is an ideal way to group similar files and compress large files to reduce their size. The compression is lossless. This means that using the compression algorithm, we can operate on the compressed data to perfectly reconstruct the original data. So, in Python Zipfile is an archive file format and a compression standard; it is a single file that holds compressed files.

Benefits of Python Zipfiles

Bunching files into zips offer the following advantages:

1. It reduces storage requirements

Since ZIP files use compression, they can hold much more for the same amount of storage

2. It improves transfer speed over standard connections

Since it is just one file holding less storage, it transfers faster **(4 Marks)**

Reading ZIP Files

To read the contents of a ZIP file, we can use the ZipFile class from the built-in zipfile module in Python.

First, import the zipfile module. Then, we can create a new instance of the ZipFile class and specify the name and path of the zip file that we want to read. After that, we can use the methods provided by the ZipFile class to access the contents of the zip file.

The ZipFile class provides several methods for reading zip files, such as namelist() , infolist() , read() , and extract() . The namelist() method returns a list of all the file names in the zip archive, while the infolist() method returns a list of ZipInfo objects for all the files in the archive.

The read() method can be used to read the contents of a specific file in the zip archive, while the extract() method can be used to extract a specific file from the archive and save it to a specified location Zip file 3 on the filesystem. **(3 Marks)**

When reading a zip file in Python, it is important to be aware of the file paths and naming conventions used in the archive, as these may differ from the paths and naming conventions used on the local filesystem. In addition, some zip archives may be password-protected or encrypted, which may require additional steps to read or extract the contents of the archive.

Code:-

```
With ZipFile('aa.zip') as myzip:
 with myzip.open('ac.txt') as myfile:
     print(myfile.readline())
```

(3 Marks)

7.c) List out the differences between shutil.copy( ) and shutil.copytree( ) method.(6 Marks)

7.c) While shutil.copy() will copy a single file, shutil.copytree() will copy an entire folder and every folder and file contained in it. Calling shutil.copytree(source, destination) will copy the folder at the path source, along with all of its files and subfolders, to the folder at the path destination. The source and destination parameters are both strings. The function returns a string of the path of the copied folder. **(2 Marks)**

Code: -

```
import shutil, os
os.chdir('C:\\')
shutil.copytree('C:\\bacon', 'C:\\bacon_backup')
'C:\\bacon_backup'
```

(4 Marks)

Q 8a) Briefly explain assertions and raising an exception. [6 marks]

8a) **Assertions**

An assertion is a sanity check to make sure your code isn't doing something obviously wrong. These sanity checks are performed by assert statements. If the sanity check fails, then an AssertionError exception is raised.

In code, an assert statement consists of the following:

The assert keyword

A condition (that is, an expression that evaluates to True or False)

A comma

A string to display when the condition is False

For example, enter the following into the interactive shell:

```
>>> ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73]
>>> ages.sort()
>>> ages
[15, 17, 22, 26, 47, 54, 57, 73, 80, 92]
>>> assert
ages[0] <= ages[-1] # Assert that the first age is <= the last age.
```

The assert statement here asserts that the first item in ages should be less than or equal to the last one. This is a sanity check; if the code in sort() is bug-free and did its job, then the assertion would be true. **(3 Marks)**

Raising Exceptions

Python raises an exception whenever it tries to execute invalid code. One can also raise his own exceptions in your code. Raising an exception is a way of saying, "Stop running the code in this function and move the program execution to the except statement."

Exceptions are raised with a raise statement. In code, a raise statement consists of the following:

The raise keyword

A call to the Exception() function

A string with a helpful error message passed to the Exception() function

If there are no try and except statements covering the raise statement that raised the exception, the program simply crashes and displays the exception's error message.

Often it's the code that calls the function, rather than the function itself, that knows how to handle an exception. That means you will commonly see a raise statement inside a function and the try and except statements in the code calling the function. For example:

Raise a TypeError if x is not an integer:

```
x = "hello"
if not type(x) is int:
  raise TypeError("Only integers are allowed")
```
(3 Marks)

8 b) List out the benefits of using logging module with an example.

8b) To enable the logging module to display log messages on your screen as your program runs, copy the following to the top of your program (but under the #! python shebang line):

import logging

logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)

s - %(message)s') **(2 Marks)**

Basically,

when Python logs an event, it creates a LogRecord object that holds information about that event.

The logging module's basicConfig() function lets you specify what details about the LogRecord object you want to see and how you want those details displayed. **(2 Marks)**

Say you wrote a function to calculate the factorial of a number. In mathematics, factorial 4 is 1 × 2 × 3 × 4, or 24. Factorial 7 is 1 × 2 × 3 × 4 × 5 × 6 × 7, or 5,040. Save the program as factorialLog.py.

import logging

```
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s -  %(levelname)s
-  %(message)s')
logging.debug('Start of program')
def factorial(n):
    logging.debug('Start of factorial(%s%%)'  % (n))
    total = 1
    for i in range(n + 1):
        total *= i
        logging.debug('i is ' + str(i) + ', total is ' + str(total))
    logging.debug('End of factorial(%s%%)'  % (n))
    return total

print(factorial(5))
logging.debug('End of program')
```

(3 Marks)

The benefit of logging levels is that you can change what priority of logging message you want to see. Passing logging.DEBUG to the basicConfig() function's level keyword argument will show messages from all the logging levels (DEBUG being the lowest level). But after developing your program some more, you may be interested only in errors. In that case, you can set basicConfig()'s level argument to logging.ERROR. This will show only ERROR and CRITICAL messages and skip the DEBUG, INFO, and WARNING messages. **(1 Marks)**

8 c) Develop a program with a function named DivExp which takes two parameters a, b and returns a value C (C=a/b). Write a suitable assertion for a >0 in function DivExp and raise an exception for, when b = 0. Develop a suitable program which reads two values from the console and calls a function DivExp.(8)

8c)

```
def DivExp(a,b):
```

```
    # AssertionError with error_message.

    assert  a>0, "Value of a should be greater than 0" # denominator can't be 0
    if b==0:
        raise Exception('Denominator should be greater than 0')
    c=a/b
    return (c)

a=int(input("Enter 1st no:"))
b=int(input("Enter 2nd no:"))
res=DivExp(a,b)
print("The result is:",res)
```

(8 Marks)

Q.9 a) Define a class and object, construct the class called rectangle and initialize it with height =100, width = 200, starting point as (x = 0, y =0). Write a program to display the center point coordinates of a rectangle. (8)

9a)

```
class Point:
""" This is a class Point
representing coordinate point
"""
class Rectangle:
""" This is a class Rectangle.
Attributes: width, height and Corner Point
"""
def find_center(rect):
p=Point()
p.x = rect.corner.x + rect.width/2
p.y = rect.corner.y + rect.height/2
return p
```

```
def resize(rect, w, h):
rect.width +=w
rect.height +=h
def print_point(p):
print("(%g,%g)"%(p.x, p.y))

box=Rectangle() #create Rectangle object
box.corner=Point() #define an attribute corner for box
box.width=100 #set attribute width to box
box.height=200 #set attribute height to box
box.corner.x=0 #corner itself has two attributes x and y
box.corner.y=0 #initialize x and y to 0

print("Original Rectangle is:")
print("width=%g, height=%g"%(box.width, box.height))
center=find_center(box)
print("The center of rectangle is:")
print_point(center)
resize(box,50,70)
print("Rectangle after resize:")
print("width=%g, height=%g"%(box.width, box.height))
center=find_center(box)
print("The center of resized rectangle is:")
print_point(center)
```

A sample output would be:

Original Rectangle is: width=100, height=200

The center of rectangle is: (50,100)

Rectangle after resize: width=150, height=270

The center of resized rectangle is: (75,135) **(8 Marks)**

9b) Explain the concept of copying using the copy module with an example.(6)

9b) Copy():The copy() method of the copy module duplicates the object. The content (i.e. attributes) of
one object is copied into another object as we have discussed till now. But, when an object itself is an attribute inside another object, the duplication will result in a strange manner.

```
import copy
class Point:
""" This is a class Point
representing coordinate point
"""
class Rectangle:
""" This is a class Rectangle.
Attributes: width, height and Corner Point
"""
box1=Rectangle()
box1.corner=Point()
box1.width=100
box1.height=200
box1.corner.x=0
box1.corner.y=0
box2=copy.copy(box1)
print(box1 is box2) #prints False
print(box1.corner is box2.corner) #prints True
```

Now, the question is – why box1.corner and box2.corner are same objects, when box1 and box2 are different? Whenever the statement

box2=copy.copy(box1) **(3 Marks)**

Deepcopy():

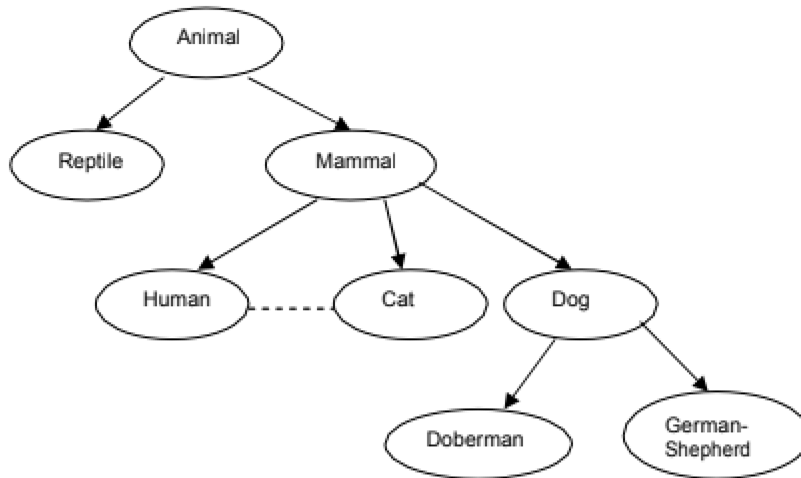
Python provides a method deepcopy() for creating two independent physical objects. This method copies not only the object but also the objects it refers to, and the objects they refer to, and so on.

```
box3=copy.deepcopy(box1)
print(box1 is box3) #prints False
print(box1.corner is box3.corner) #prints False
```

Thus, the objects box1 and box3 are now completely independent. **(3 Marks)**

9c) Explain the concept of inheritance with an example.

9c) Inheritance is a process by which one object can acquire the properties of another object. It supports the concept of hierarchical (top-down) classification. For example, consider a large set of animals having their own behaviors. In that, mammals are of one kind having all the properties of animals with some additional behaviors that are unique to them. Again, we can divide the class into various mammals like dogs, cats, humans etc. Again among the dogs, differentiation is there like Doberman, German-shepherd, Labrador etc. Thus, if we consider a German-shepherd, it is having all the qualities of a dog along with its own special features. Moreover, it exhibits all the properties of a mammal, and in turn of an animal. Hence it is inheriting the properties of animals, then of mammals and then of dogs along with its own specialties. We can depict it as shown in the Figure given below. **(3 marks)**



Example of Inheritance

Normally, inheritance of this type is also known as “is-a” relationship. Because, we can easily say “Doberman is a dog”, “Dog is a mammal” etc. Hence, inheritance is termed as Generalization to Specialization if we consider from top-to-bottom level. On the other hands, it can be treated as Specialization to Generalization if it is bottom-to-top level. This indicates, in inheritance, the topmost base class will be more generalized with only properties which are common to all of its derived classes (various levels) and the bottom-most class is most specialized version of the class which is ready to use in a real-world.

If we apply this concept for programming, it can be easily understood that a code written is reusable. Thus, in this mechanism, it is possible for one object to be a specific instance of a more general case. Using inheritance, an object need only define those qualities that make it unique object within its class. It can inherit its general attributes from its parent.**(3 marks)**

10.a) Define a function which takes two objects representing complex numbers and returns new complex number with a addition of two complex numbers. Define a suitable class Complex' to represent the complex number. Develop a program to read NN>=2) complex number S and to compute the addition of N complex numbers. (8Marks)

10 a) Code:-

```
class Complex:

    # Constructor to accept
    # real and imaginary part
    def __init__(self, tempReal, tempImaginary):
        self.real = tempReal;
        self.imaginary = tempImaginary;

    # Defining addComp() method
    # for adding two complex number
    def addComp(self, C1, C2):

        # creating temporary variable
        temp=Complex(0, 0)

        # adding real part of complex numbers
        temp.real = C1.real + C2.real;

        # adding Imaginary part of complex numbers
        temp.imaginary = C1.imaginary + C2.imaginary;

        # returning the sum
        return temp;

# Driver code
if __name__=='__main__':

    # First Complex number
    C1 = Complex(3, 2);

    # printing first complex number
    print("Complex number 1 :", C1.real, "+ i" + str(C1.imaginary))

    # Second Complex number
    C2 = Complex(9, 5);

    # printing second complex number
    print("Complex number 2 :", C2.real, "+ i" + str(C2.imaginary))
```

```
# for Storing the sum
C3 = Complex(0, 0)

# calling addComp() method
C3 = C3.addComp(C1, C2);

# printing the sum
print("Sum of complex number :", C3.real, "+ i"+ str(C3.imaginary))
```

(8 Marks)

Output

Complex number 1 : 3 + i2

Complex number 2 : 9 + i5

Sum of complex number : 12 + i7

10.b) Explain __init__() and __str__() method with examples.(6 Marks)

10.b)

Solution: Each method with explanation and program

In Python, __init__() and __str__() are two special methods that are commonly used in classes.

__init__() is a constructor method in Python that is used to initialize the object's attributes. It is called when an instance of the class is created. The self parameter in __init__() refers to the instance of the class that is being initialized, and can be used to set attributes for that instance.

__str__() is a special method in Python that is used to define a string representation of an object. It is called when the str() function is called on an object. The self parameter in __str__() refers to the instance of the class, and can be used to access its attributes and return a string representation of the object.

Here is an example program that demonstrates the use of __init__() and __str__() methods in Python:

```
class Person:
    def __init__(self, name, age):   [ 3 marks]
        self.name = name
        self.age = age
    def __str__(self):    [ 3 marks]
        return f"{self.name} ({self.age} years old)"
        person1 = Person("John", 30)
        person2 = Person("Jane", 25)
        print(person1) # Output: John (30 years old)
        print(person2) # Output: Jane (25 years old)
```

In this program, we define a Person class that has two attributes: name and age. We use the __init__() method to initialize these attributes when an instance of the class is created.

We also define the __str__() method to return a string representation of the object. When we print an instance of the Person class using the print() function, the __str__() method is called to return a string representation of the object.

10.c) Briefly explain the printing of objects with an examples.(6)

10.c) An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances. **(2 Marks)**

Printing objects give us information about the objects we are working with. In C++, we can do this by adding a friend ostream& operator << (ostream&, const Foobar&) method for the class. In Java, we use toString() method. In Python, this can be achieved by using __repr__ or __str__ methods. __repr__ is used if we need a detailed information for debugging while __str__ is used to print a string version for the users.

Code:- **(4 Marks)**

```
# Defining a class
class Test:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __repr__(self):
```

```
        return "Test a:% s b:% s" % (self.a, self.b)

    def __str__(self):
        return "From str method of Test: a is % s, " \
               "b is % s" % (self.a, self.b)

# Driver Code
t = Test(1234, 5678)

# This calls __str__()
print(t)

# This calls __repr__()
print([t])
```

Output:-

From str method of Test: a is 1234, b is 5678

[Test a:1234 b:5678]